Surrey GU24 9BX (GB). MILLS, Robert, Ian [GB/GB]; 9 Southern Hay, Hartley Wintney, Hampshire RG27 8TZ (GB). JIN, Yih-Chyun [CN/GB]; 43 Curzon Road, London W5 1NE (GB).

(74) Agent: LANGLEY, Peter, James; Origin Limited, 52 Muswell Hill Road, London N10 3JR (GB).

[Continued on next page]

(54) Title: A PRESENTATION SERVER WHICH ENABLES A CLIENT DEVICE TO RUN A NETWORK BASED APPLICATION

Overview of the components of the ALC
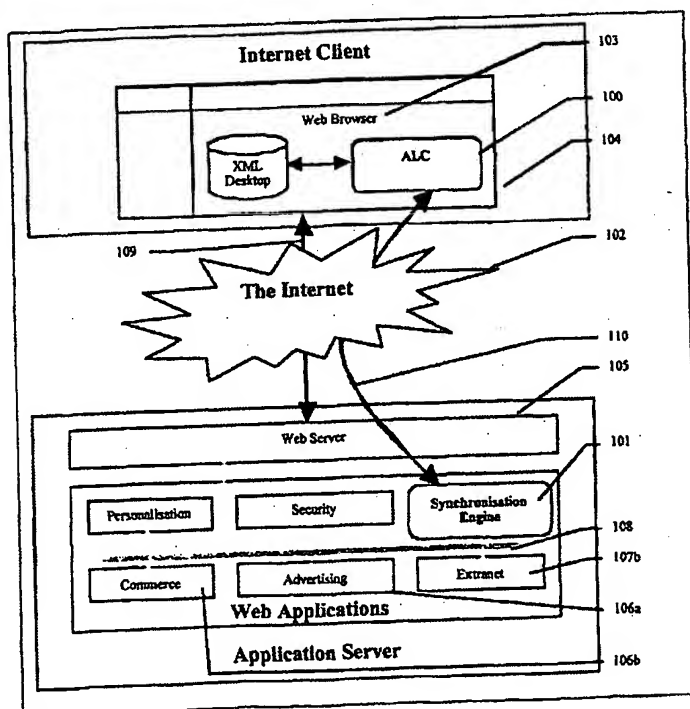
(57) Abstract: A presentation server which enables a client device to run a network based application by sending structured data from one or more data sources to software on the client device. The presentation server decouples the client device from the data source by (i) insulating the data source from needing to handle directly any display, data storage, data querying or communications functions relating to the client device and (ii) insulating the client device from needing to handle communications functions which are specific to the data source.

WO 02/065278 A2

European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

A PRESENTATION SERVER WHICH ENABLES A CLIENT DEVICE TO RUN A NETWORK BASED APPLICATION

BACKGROUND TO THE INVENTION

5

1. Field of the invention

This invention relates to a presentation server which enables a client device to run a network based application client software enabling a client to run a network based application. It

10 finds particular utility in enterprise computing. A 'presentation server' is generally any kind of server that determines how data will be displayed on a client device.

2. Description of the Prior Art

15 Enterprise computing was originally conducted through large mainframe computers. These mainframes gave the end-user very limited application functionality, displaying only text messages and requiring keyboard strokes for every command (for example, there was no mouse and none of the richness of today's desktop applications like drag and drop, graphing, sorting or personalized settings). Deployment costs were also very high due to the cost of

20 hardware and the lack of skilled computer operators.

The evolution to client/server-based computing provided much richer functionality. The graphical user interface (GUI) provided end-users a much simpler, more efficient method to interact with applications, but systems still proved expensive and difficult to maintain. In

25 theory, applications would be deployed on the server centrally and distributed via the network to each of the clients. But business users had different needs of functionality and therefore required different applications with unique configurations. Not only did some information have to be personalized, but information also had to be retained on the desktop. This meant it was not possible to "write once and deploy many". The result was that the

2

end-user got the desired rich functionality, but much of the code for the business logic for the applications was located on individual clients. IT Departments' budgets ballooned due to high support costs and slow implementation times. Rolling out new applications – or even minor updates – required a unique installation for every PC that required access.

5

The advent of the Internet was supposed to change all this. The Internet provided an infrastructure that lent itself to the notion of zero-cost deployment with the promise of "write once, run anywhere". Such a model would significantly decrease the cost of deploying enterprise systems. With the Internet, back-end servers would run the application centrally,

10   and all that users required to access the application was a browser. There would be no code specific to the application running on the user's client. The potential this promised to large companies was unprecedented.

Despite the enormous potential, the initial technology (HTML) used for building and

15   deploying the front-end of these new Internet applications was never intended for anything more than displaying simple text. As companies tried to deploy sophisticated business systems over the Internet, they soon realized that much of the desired functionality they wanted was not feasible. HTML based Web Applications did not have true interactivity or access to data in real-time. Business users needed to interact with and manipulate the data,

20   but mostly what they saw was static page views. Certain solutions were designed to provide a more interactive user interface. However, they still used variations of HTML, resulting in passive documents with numerous limitations.

Specifically, from a technical perspective these limitations resulted in:

25

• **Lack of functionality:** No matter how sophisticated the back-end business logic, end-users lost the functionality they had come to expect from their familiar client/server-based system. Large online systems that were deployed severely lacked the rich functionality that companies demanded in their offline systems. While lots of money was spent on building

these systems, to date they have clearly not replaced client/server systems. In fact, very few even augment the existing offline systems, and in retrospect proved to be very expensive IT exercises which fell short of their original promise. There have been early attempts at providing a richer user interface and a more robust connection between the back-end servers and the client by using JavaScript, DHTML, and various plug-ins. However, these solutions present expensive development cycles and ongoing maintenance, while introducing new security issues (see below), and applications that still fall short of the desired functionality.

• **Load on back-end Servers:** The HTML-based Web represents a very inefficient use of computing resources. Publishing HTML's required page by page structure means that as more people log on to a system, requests to the database and the application server put a strain on these servers. Companies needing to scale operations can buy more servers with more CPUs to handle the high load, but each additional processor not only costs a lot, but may also affect the license cost of all the various software packages running on the server.

• **Security Concerns:** Companies are concerned that data pushed beyond corporate firewalls onto the Internet is open to unauthorized, third-party scrutiny. Additionally, there is concern that plug-ins downloaded from various Internet applications may contain viruses. Most software companies license their software on a per-CPU basis. As companies add more CPUs to their servers or add entirely new servers, the licensing fee for each piece of software increases by a multiple of how many new processors are added.

• **Cannot Easily Prototype New Features:** Given the complexity of developing user interfaces that provide adequate functionality and the effort required to integrate such solutions to the back-end, companies typically do not prototype and roll out new features incrementally. Very often, further development of these systems is an extensive project, which introduces the uncertainty of never being fully deployed and used. Because of the effort required for each release, many companies shy away entirely from going forward on certain projects as opposed to rapidly developing incremental pieces to see what parts work.

From a business manager's perspective, these limitations meant:

• **Inadequate Systems for Doing Business:** Client/server-based systems were expensive,
difficult to build and deploy and not accessible outside of the enterprise. However, they do
provide the functionality needed to actually do business, whether for trading financial
instruments, managing resources and accounts through ERP systems or selling and
maintaining client relationships through SFA and CRM tools. A fraction of these capabilities
are available over the Internet in certain systems, but are far from giving the functionality
people require to be effective.

• **Slow Performance:** Sites using HTML force users to click and wait for a response while
the back-end system processes the information and pushes a new page to the users. This can
take a few seconds at best, or tens of seconds, even minutes, at worst. Users of these systems
simply abandon activities when they take too long or when connections entirely fail.

• **Expensive Development:** Building and deploying enterprise-level systems over the
Internet with sophisticated functionality requires a lot of custom coding with long and
expensive development cycles. Most of this work cannot be leveraged for other systems and
is a large, one-off expense that companies must absorb.

• **Expensive Maintenance:** Complex Web systems contain thousands of lines of code on
both the back-end (business logic) and the front-end (delivers the interface). The connection
between the front-end and backend also contains thousands of lines of custom code which
needs to be debugged and maintained. All changes to the code require programmers to
manually write new code and test the functionality through staging before going to a live
production site.

• **Expensive Bandwidth:** In today's online systems, information is presented to users

through entire Web pages, which are pushed out to users over the network. As information changes or transactions are completed, users can request new pages or submit pages back to the system. However, every time pages are sent across the network, companies pay for the bandwidth used. As online systems add users and these users start to do more activities online, they consume more and more bandwidth, and as a result costs increase quickly.

• **Mobile Access**: As the reach of the Internet extends beyond the PC, enabling mobile access (i.e. PDAs, Cellular Phones) to business applications is becoming a standard business requirement. Current solutions for wireless delivery require separate application re-writes (and expertise) for each device supported.

• **Hiring and Retraining expensive IT Staff**: Integration of Web systems with existing offline systems and the maintenance of more sophisticated user interfaces for the Web using JavaScript and HTML typically requires armies of programmers, burdening the budget of every department, and many times putting the overall company's viability in jeopardy.

As a result of these challenges experienced to date, companies still have limited confidence in deploying their systems through the Web. Re-creating the same feature-rich interfaces that existed on client/server-based desktop applications has proven futile for companies attempting to move their systems to the Internet. Replicating the performance of desktop applications as well as standardizing the collaboration and interaction between disparate Web applications has been even more complex, expensive, and mostly elusive. The massive trade-off in end-user productivity, combined with security, scalability and performance concerns, has resulted in enterprises running client/server systems in parallel with Web-enabled systems, further deepening the cost of IT, and still not realizing most of the benefits the Internet promised.

Hence, even though millions of people are now using the Internet to look at documents from anywhere in the world, true interactivity to manipulate data is missing, and the

evolution of the Web into a mainstream business and trading environment simply has not happened.

SUMMARY OF THE PRESENT INVENTION

In a first aspect of the invention, there is a presentation server which enables a client device to run a network based application by sending structured data from one or more data sources to software on the client device, wherein the presentation server decouples the client device from the data source by (i) insulating the data source from needing to handle directly any display, data storage, data querying or communications functions relating to the client device and (ii) insulating the client device from needing to handle communications functions which are specific to the data source.

Because the present invention relates principally to a client server architecture, it may be helpful to describe the entire client/server architecture at this point so that the presentation server ('Presentation Server') can be seen in context. This client server architecture envisages a new approach which is neither entirely thick client, nor entirely client-server, nor entirely fully distributed, but blends aspects of each. (The Presentation Server could in principle also be implemented in a peer to peer architecture, or in an entirely closed device, but the client server architecture is currently seen as the most commercially significant).

The Presentation Server receives structured data (or super-sets of it) over a network from an originating data source, which may be a web server, a back end application, a messages queue or a web service (or multiple sources of these types) able to generate structured data, such as XML or SQL format data directly (or indirectly through a generator). It then intelligently passes that data onto client devices which require it. The client in one implementation is a unique three layer client, in which the three layers are:

        (a)   a communications layer to send and receive messages over the network;

        (b)   a database layer to store, and allow querying of, the structured data;

        (c)   a rendering layer which generates, from the structured data in the database layer, data for a user interface;

such that the client software is self-contained to provide all of the communications, data storage and rendering resources needed to run the network based application on the client device.

5   The Presentation Server subsequently keeps the database layer in the '3 Layer System' automatically updated (or 'synchronised') with the originating data source. The Presentation Server insulates the originating source from needing to know which clients need to be updated, what updates they need, how to communicate with clients, and the user interface at the clients. This overhead is instead handled by the Presentation Server; this approach

10 allows a networked application developer to concentrate on application business logic development, rather than the front-end display/communications and synchronisation elements that traditionally take over 50% of developers' time. Strictly separating the data source from the client device allows the business logic of an application to be fully independent of the presentation logic on the client device, such that the data source need not

15 even know of the existence of a client device even though it may be a necessary part of the network application. Similarly, it allows the presentation logic of a client device to be fully independent of the business logic on the data source, even to the extent that the client device need not even know of the existence of a data source even though it may be a necessary part of the network application.

20

By pre-fabricating these complex functions and capabilities into a generic Presentation Server, networked application developers can produce finished applications far more quickly than previously. Scaling to large numbers of clients (10000s) is also possible, even for legacy back end systems which are limited to serving 10 or fewer clients. This is because the back

25 end system needs to supply information just once to the Presentation Server; the Presentation Server itself is fully scalable (e.g. multiple CPUs; multiple CPUs on multiple machines etc.) to serve the required numbers of client devices.

Communication between the Presentation Server and the data source can be carried out locally or over a network and interaction can occur:

- via services provided by the data source (e.g. HTTP & CGI; SOAP);
- via message queues to which the data source publishes and the Presentation Server subscribes;
- via message queues to which the Presentation Server publishes and the data source subscribes;
- via direct calls to the data source's application programmable interface (API) where the underlying system provides the low-level protocol (e.g. Remote procedure calls; CORBA);
- via direct calls to one or more databases on the data source;
- in a loosely coupled way (e.g. via e-mail or ftp file transfer).

The Presentation Server can be thought of as allowing the data source to configure or handle *indirectly* any display, data storage, data querying or communications functions relating to the client device by forwarding communications from the data source to client software on the client device. The data source only indirectly configures the client device because it is the Presentation Server which:

- manages updates of the structured data held on the client device by regularly polling the data source for changes;
- directly manages distribution of streaming data to multiple client devices, where the data source needs only to provide a single stream to a Presentation Server which in turn broadcasts the information in a scalable fashion to multiple (potentially more than tens of thousands) of client devices that may in turn be connected to the Presentation Server at different performance rates over the network;
- handles communications functions specific to different client devices by having a generic communications capability (data receipt/acknowledgement/response) enabling it to communicate with multiple client browsers, multiple client device types and across multiple bearers; or

- handles communications functions specific to different client devices by having a generic display capability by being able to download to the database layer one or more of the following, each configured appropriately for a destination display, device type and bearer: such as panels, windows, controls, data binding to the controls and event models.

5

Additionally, the Presentation Server can insulate the client device from:

- needing to directly handle communications functions which are specific to the data source, by translating messages from the client device to a form suitable for the data source;

10

- needing to directly interpret raw data provided by the data source, by translating messages from the data source to a form suitable for the client device;

- needing to directly manage updates of the structured data held on the client device, by receiving a single data update from the data source and being solely responsible for identifying all client devices that need that data update, and sending the appropriate updating data to all of the client devices that need it;

15

- needing to know the quantity or configuration of multiple data sources to which the Presentation Server is connected, by aggregating data from multiple data sources into a single data stream for each client device.

20

Further key features of implementations of the Presentation Server are defined in the claims. Implementations of the invention include a Presentation Server which:

- maintains a data stream per client device in which the data is pushed over a persistent connection to the client device over the network as soon as it is available;

25

- maintains a data stream per client device in which the data is sent to the client device over the network in response to regular or ad-hoc polling requests from the client device;

- compresses the data received from the data source prior to sending the information to the client device;

- encrypts the data received from the data source prior to sending the information to the client device, thus insulating the data source from the enhanced security needed to allow a remote client device to run the network application over a public or insecure network;

- decrypts the data received from the client device prior to sending the information to the data source, thus insulating the data source from the enhanced security needed to allow a remote client device to run the network application over a public or insecure network;

- can process a request by the client device for an update to a data source and return an indication of whether the update was successful as indicated by a response from the data source; if the update fails, the data source may optionally provide further information about the reasons for the failure in a way that can be displayed or interpreted by the client device (e.g. in an equity trading application, the data source might indicate that the quantity of a stock to be purchased exceeds some limit imposed by business rules and can notify the Presentation Server of the problem who in turn can forward the information to the client device);

- can process a request by the client device for an update to a data source and return an indication of whether the update was successful even if the data source does not directly respond with a success or failure indication (e.g. where no response for a period implies a failure to update the data source); it may be configured to automatically generate responses to the client device on behalf of the data source;

- can process a request by the client device for a single transaction update to multiple data sources and return an indication of whether the update was successful;

- can process a request by the client device for a single transaction update to multiple data sources in such a way that all data source updates are only confirmed once each data source has preliminarily indicated that the update would succeed;

- provides that failure of any one of the transaction updates causes all the transaction updates to be cancelled and a failure response to be returned to the client device that requested the transaction;

- provides that a preliminary validation of all transactions causes all transactions to be confirmed and returns a success indication to the client device that requested the transaction;
- returns the message that results in the user interface at the client device being updated at the same time as it issues a message that results in the user interface of different client devices being updated in the same manner;
- can update marginal portions of the database layer in a client device by sending, instead of an entire structured data tree hierarchy, a key which identifies the part of the structured data tree which has changed;
- can update a database layer by altering, in real time, configuration data held in the database layer which defines how the structured data can be interacted with from within the user interface;
- can prioritise updates to a client device that is returning on-line after an absence;
- can update the database layer by re-initialising all of the structured data on a client device that is returning on-line after an absence by applying rules to determine if re-initialisation is appropriate;
- stores updated structured data in a transaction log record from which a client can pull updates;
- stores updated data in a transaction log record and pushes updates as appropriate to the client device;
- is able to send updating data to different client devices at different times or frequencies;
- can be configured to skip unnecessary intermediate updates to streaming data for devices whose connection or client device is too inefficient to receive and process every update (e.g. an equity quote application might update 10 times a second, but a certain device might not accept more than an update every 5 seconds. In this case, the Presentation Server would effectively send only every 50th update to that client device, whilst it might simultaneously send all 50 updates to another more efficient client device);

- can maintain a statefull connection with the client device, e.g. by routing a unique, incrementing number to the client device, which the Presentation Server logs in relation to all data updates, so that when the client device returns on-line from an absence, the client device sends the latest number it received back to the Presentation Server so that the Presentation Server can determine what updates the client device did not receive and therefore need to be sent;

- is sent by the client device any parameters that need to be persistent and stores these parameters or routes them to storage for later retrieval;

- provides scalability in the number of client devices a network based application can communicate with by being itself scalable with:

  (a) multiple CPUs on a machine;

  (b) multiple CPUs on multiple machines in a parallel or in a parent/child arrangement;

  and manages the message flows from each CPU to the applicable client devices;

- provides fault tolerance by being itself scalable with:

  (a) multiple CPUs on a machine;

  (b) multiple CPUs on multiple machines in a parallel or in a parent/child arrangement, providing a mechanism to automatically compensate for failures of instances on one or more CPUs by reconfiguring the network configuration at run-time and optionally re-establishing connectivity for each orphaned client device to an alternate Presentation Server;

- can form part of a network of several geographically separated Presentation Servers, some or all of which operate as edge servers;

- can form part of a network of several geographically separated machines organised into a topology which is appropriate to the physical arrangement of sites in a business running the network based application;

- can combine structured data from several different server based applications for a given client device;

- can determine automatically and in real time, when a client device requests several images, whether it should reduce bandwidth consumption by placing all of the images into a single, composite image file, each image sharing data which is not image specific;

5
- modifies the data it sends to a client in dependence on information sent to it from the client which relates to the display limitations of the client device.

- uses a session object associated with an applet to determine the specific display parameters and structured data to be served to the client device.

10   Other aspects of the invention include:

- A web application which, when invoked or run, causes a Presentation Server as defined above to be run;

15
- A hardware server which hosts a Presentation Server as defined above.

20

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be described with reference to the accompanying drawings, in which:

5

Figure 1 is a schematic of the main components of the AltioLive Client (ALC);

Figure 2 is a schematic showing the order of events, which occur when a client communication is initiated;

10

Figure 3 is a schematic showing the chain of events, which occur when a user does something, which requires additional data;

Figure 4 is a schematic of various aspects of the ALC configuration;

15

Figure 5 is a schematic of a simple Synchronization Engine configuration;

Figure 6 is a schematic of a complex Synchronization Engine configuration;

20   Figure 7 is a schematic of the AltioLive Deployment Architecture;

Figure 8 is a representation of a screen view. The initial data served by the Synchronization Engine is shown in this example;

25   Figure 9 is a representation of a Service Function;

Figure 10 is a representation of a screen view depicting the Datapool;

Figure 11 is a representation of a screen view of a "NEW_SALE" Service Function;

Figure 12 is a representation of a screen view of a "NEW_BID" Service Function;

Figure 13 is a representation of a "Select Product" list, including a GET_APPIMG image

5   server command;

Figure 14 is a schematic of the full system architecture of an AltioLive application;

Figure 15 is a schematic of the interaction between the Synchronization Engine and the

10   Application with emphasis on three main areas; the Data Service Function, Image Service

Functions and Datapools;

Figure 16 is a schematic of AltioLive in Dynamic Mode. In Dynamic mode it is necessary to

change the HTML to reflect the connection to the Synchronization Engine;

15

Figure 17 is a schematic of the Logon process used by the AltioLive Demonstration

applications.

DETAILED DESCRIPTION OF THE PREFERRED IMPLEMENTATION

The invention will be described with reference to an implementation called AltioLive, from Altio Inc. of Boston, Mass., USA. This Detailed Description is organized as follows:

5

A.   AltioLive – a Brief Overview of the Overall Architecture

B.   AltioLive Client – Overview

C.   AltioLive Presentation Server – Overview

D.   AltioLive Client – Details

10   E.   AltioLive Synchronisation Engine (SE) – Details

F.   Deploying AltioLive

G.   Security Aspects of AltioLive

H.   AltioLive – Benefits of using AltioLive

I.   Glossary

15

Appendix I   "Getting Started with AltioLive" from Altio Inc.; Chapter 3 titled 'The AltioLive Presentation Server'

Appendix II   "Integrating AltioLive" from Altio Limited; Volume 4 Chapters 1- 5 of the AltioLive User Documentation.

20

A. AltioLive – a Brief Overview of the Overall Architecture

AltioLive solves the technical challenges of streamlining development, deployment and

25   maintenance of real-time interactive Internet/web applications. Updates to data items are received 'live' at a client device and can be transmitted concurrently and automatically to client devices to avoid users requesting updates and then having to wait for server responses. Access to the application through alternate devices (e.g. wireless access) presents the user with their data automatically rendered for display and navigation on the device, whether it is

a cell phone or a Personal Digital Assistant (PDA). For certain PDAs, AltioLive can be implemented directly (e.g. IPAQ). For cell phones, Web TV and lightweight PDAs, the application is provided via HTML or WML pages generated by a renderer engine in the Presentation Server. Thick client functionality is not provided on these devices.

5

Applications developed with AltioLive provide users the ability to move between windows inside the browser, scroll or sort through lists, resize columns and see details on selected items (including charts and graphs). The windows can be minimized, maximized, moved around and resized, just like in a standard desktop application. Customizable toolbar buttons

10   allow the user to pop-up additional windows to view further details such as lists, forms or graphs. Data entry is provided through the typical controls such as text boxes, checkbox, drop-down lists etc. with additional complex controls such as graphs, tree-views, date and color pickers.

15   Since developing applications with AltioLive requires virtually no coding, development and ongoing maintenance efforts are reduced up to 75% while bandwidth and server-side processing decrease by a similar order of magnitude.

In AltioLive, an Altio Presentation Server ('APS') deploys and intelligently manages real-

20   time, interactive Internet or Web applications by serving appropriate XML data to client devices. The functions of the APS are (a) to receive and store XML data from the source of that data (e.g. web server, web service provider etc.); (b) to supply to each client device an appropriately configured 3 Layer System - in this instance a generic applet (as described above) — this generic applet handles client device display and communications functions; (c)

25   configuration, initial data and supporting files (images and 'skins') and (d) to provide XML formatted data updates to the client in real-time and (e) to process client device requests to process updates and /or provide further information.

The APS therefore assumes responsibility for properly configuring the generic applet so that the database layer (an XML database layer in the case of AltioLive) on the client displays the correct data for a particular user, in the correct format and with window controls appropriate to that user, and in an optimal manner for the display capabilities of different client devices.

5   It also assumes responsibility for all communications functions to/from the client device by appropriately configuring the generic applet, which it sends to the client device. The communications functions enable data to be exchanged with any device type and OS (e.g. PC, Apple™, Unix™, Symbian™, Palm™ etc) and over any bearer/protocol.

10   The updating function ensures that all client devices are up to date or synchronized. In AltioLive, this function is achieved using one or more clustered 'Synchronization Engines' (SE). A SE is a server-side software application, which runs on the Altio Presentation Server, and coordinates and controls the interaction between multiple clients, Web applications and server data sources over multiple communication protocol types, while still being

15   configurable via industry standard XML. The SE allows for enterprise scale real-time data updates without the need for a client to use a browser 'refresh' button; live data synchronization across multiple users is possible.   Static applications can therefore be transformed into dynamic ones and these applications can be extended to multiple devices in different formats, all from a centrally managed location.

20

In AltioLive, the generic applet is called the AltioLive Client; this applet enables live data to be displayed in fully interactive windows inside a user's browser. Upon a request to the web server or other ultimate data source, by the client device, the AltioLive Client is deployed to the user's machine from the SE as a small Java applet. The applet is generic and self-

25   contained – i.e. no additional code is needed to make it suitable for a specific purpose. Instead, it interprets the configuration file sent from the SE to implement different views and behaviours so that it works for a specific web application/client device/bearer combination.

B. AltioLive Client – Overview

The AltioLive client has three layers:

5   (a) a generic communications functions layer (establishing and maintaining connections; re-connecting; updating etc.);

(b) a generic rendering/graphics layer (generates appropriate screen responses to mouse drags and actions etc.) and

(c) a database layer (stores applet configuration parameters and UI parameters like windows,
10   window controls, data binding to these controls and event models; stores also the web application related data (a super-set of what is displayed at any time)).

All three layers are necessary parts of the 3 Layer System, and each plays an important role in delivering the desired functionality/behavior.

15

B.1.   The Communications Layer
The communications layer provides a means to robustly send and receive messages between the network based application running on the client, and the necessary supporting Presentation Server.

20

Key features:
- Establish connection with Presentation Server to obtain interface configuration information (in a current implementation this is delivered as XML)
- Establish occasional on-demand connection with the Presentation Server to obtain
25   ad-hoc information as needed by the network based application (e.g. a response to a search query)
- Establish occasional on-demand connection with the Presentation Server to deliver new and/or updated network based application data to the Presentation Server (e.g. when a user submits a form), and await a response on success/failure

- Establish regular polling connections to receive network based application data to be presented in the interface
- Establish and maintain a persistent connection to a Presentation Server if required for low-latency receipt of streaming data

5
- Establish occasional connections with the Presentation Server to ensure server is alive (functioning) if needed when no streaming data has been received for a significant period.
- Interface to the Database Layer to deliver new and updated information from the Presentation Server

10
- Interface with Database Layer to receive messages to be sent to the Presentation Server (e.g. when the network based application data is changed)

B.2   The Database Layer

The Database Layer provides a means to hold and query network based application data and

15   user interface configuration data as well as an interface to the Communications Layer and the Rendering Layer.

Key Features:
- In the generic case it receives digital information from the communications layer and

20   efficiently store it for later access on demand
- Receive structured data from the Communications Layer that may later be queried using a query language to combine and/or extract specific subsets of data on demand
- The structured data may be XML; XML querying approaches can be used to combine and/or extract data subsets

25
- The Database Layer can provide temporary cacheing of network based application data on behalf of the rendering layer, in order to reduce and/or minimize the need for the 3 Layer System to communicate with the server in the course or running the network based application.

- The Database Layer supports dynamic additions, modifications and deletions to the information it holds, triggered either by the Communications Layer or the Rendering Layer.

- The Database Layer responds on demand to requests for information from the Rendering Layer.

- The Database Layer may in response to a request from the Rendering Layer, issue a request in turn to the Communications Layer to retrieve further information from the Presentation Server (e.g. when the Rendering Layer has asked for data not yet present in the Database Layer)

- The Database Layer may have a means to manage its size and may choose to expire and/or archive old or infrequently accessed information

- The Database Layer works in conjunction with the Rendering Layer to automatically trigger changes to the user interface to reflect the current state of the data in the Database Layer

B.3   The Rendering Layer

The Rendering Layer provides a means to (a) combine information describing an interface for a network based application with information to be rendered within that interface and (b) dynamically construct and manage the display of the application interface on the host client platform (e.g. a web browser or handheld device).

- The Rendering Layer generates a user interface that reflects the state of the information in the Database Layer

- Subject to the capabilities of the client device, the Rendering Layer can generate a dynamically updated user interface that continually reflects the latest state of the information in the Database Layer

- The Rendering Layer may support interaction with an end user through inputs from an external input device (e.g. a keyboard, button panel or mouse)

- The Rendering Layer may generate a network based application interface at the pixel level

- The Rendering Layer is responsible for generating and managing user interface controls (e.g. text entry boxes, charts, scroll-bars)

- The Rendering Layer may display information in a traditional desktop format using multiple overlapping views.

5 - The Rendering Layer may support user interaction with the user interface such as drag & drop of information between views, entering data into a text box, pressing a button.

- The Rendering Layer may support rendering of network based applications on devices and/or in a form other than pixels on a screen. (e.g. lines of text on a cell

10 phone or LEDs on a domestic appliance)

- The Rendering Layer may support interfaces to alternative rendering mechanisms independently of its own drawing capabilities (e.g. JavaScript calls and triggers to and from the Rendering Layer)

15 Other key features of the AltioLive Client are:

• The generic applet provides a generic display capability by being able to download from the remote Presentation Server one or more of the following, each configured appropriately for the destination browser, device type and bearer: windows, controls on

20 the windows, data binding to the controls and event models. The XML data is bound to the controls and event models, so that a web application developer need only define what data is to be associated with each window control; the applet then handles how data should be displayed when different events occur (e.g. if the data is a list of names, the applet handles how the name list is displayed if a 'sort by alphabetical order' control

25 is selected). This 'View' definition can be made specific to each user – allowing different users to be able to see or manipulate different data sub-sets and hence providing an access control mechanism.

• The applet provides a zero-client footprint in that it is (a) installed on a client device only in response to the client device initiating or requesting a web application or web service and therefore does not need to be pre-loaded or installed and, (b) after the client device has ceased using the web application or web service, it may be removed entirely from the
5   client device.

• The small size of the applet allows it to be rapidly downloaded. Once the Java applet is running on the user's machine, it maintains a persistent connection with the Presentation Server. The Presentation Server will then 'push' or propagate new data to all parties that
10   require it. Only data, which has actually changed, is pushed across the network, reducing the bandwidth used, unlike HTML systems that send an entire page when only a single data point in the page alters. Another alternative to the persistent streamed connection is a polling connection, which makes regular requests for data changes. This uses a timestamp mechanism to ensure that only new updates for the individual client making
15   the request are returned.

• The XML database provided by the applet enables XML data from two or more different web applications, web services or data sources to be combined or manipulated in one or more windows running in a browser on the client device as though from a
20   single source.

• The generic applet can generate multiple windows in a browser window of the client device, with a user able to sort data, edit data, or scroll through data in each of the windows, without the need for there to be any client side software development.

25

• The generic applet allows XML to be cut and pasted into a clipboard or other form of temporary memory. Hence, a user can drag data (e.g. numerics, text or images) from one window generated by the XML database and drop it into another window – the

appropriate changes (including computations) resulting from pasting in the new data into the applicable field of the new window are generated and displayed.

- The generic applet can generate multiple windows in a browser window of the client device, with each user able to define a personalised arrangement of windows and data to be displayed in each window, with the data to be displayed in each window capable of being obtained from several different web applications or web services.

- XML data is associated with pre-defined controls by a developer and then at run time the generic applet automatically displays the data in conjunction with some or all of those controls.

- The XML database can be accessed using XPath queries, in which standard XPath queries are enhanced with the following features: indexed retrieval; direct reference to elements and attributes identified by controls.

- Session objects can allow the remote Presentation Server to provide to the XML database the applicable data and configuration parameters appropriate to the client device and the last state of the client device known to the remote Presentation Server. This prevents unauthorised access without a session object. Session objects are user specific, so that users can access a web application irrespective of client device being used. The PS also maintains the notion of a Connection, allowing a single session to support multiple connections (two browser windows for example) each running a different view/configuration.

C. AltioLive Presentation Server (APS) - Overview

26

The AltioLive Presentation Server (APS) integrates with Web Applications to efficiently and securely distribute live XML data to users across the Internet through the range of Altio Clients. The APS has the following three functional aspects:

5   ■ Display - The APS provides a trusted architecture for the presentation of real-time interactive applications. Through a tiny, browser-based applet and XML-configuration, Internet applications deliver a completely customisable, dynamic interface with desktop-style functionality (e.g. drag and drop, resizable windows, customisable screen colours, customisable fonts, full UI configurability according to

10   user's role and preferences).

     ■ Communications - The connection between the application (on the server) and the end-user (on the client) governing the form and method of communication between the application (on the server) and display to the user (on the client). The APS

15   provides a persistent, secure connection between the application and clients, transmitting data via configurable HTTP/HTTPS polling or streaming access mechanisms.

     ■ Data Synchronization - Managing data distribution across the user-base, providing all

20   users with the latest data, in real time – from any device.

The live data is displayed in interactive windows inside the users' browsers through the appropriate client:

25   • AltioLive Client (ALC) – delivers a live, interactive, thick-client style user interface within the user's browser.

     • Pervasive Client (APC) – delivers a page-oriented user-interface rendered for user access through simple devices such as PDAs and mobiles.

• **Data Client (ADC)** – delivers live, interactive data for presentation through a DHTML interface.

5   **Synchronization Engine (SE)**
The APS is implemented as one or more clustered Synchronization Engines (SE)s that are monitored and managed through the SE Admin Tool.

10 **D.   AltioLive Client (ALC) - Details**

The AltioLive Client is a computer user interface that radically enhances how information can be displayed and updated over Internet technology.

15 An ALC is composed of a number of components:

- An internet client machine;
- A web browser supporting a Java virtual machine;
- A Java applet;
20 - An XML Desktop Database.

These components are further described below and are shown in **Figure 1.**

The ALC 100 typically fits into a configuration that consists of a Web browser 103 in which
25 the ALC runs as a small machine independent client applet. The web browser would be running on an Internet Client 104 that would be connected to a web server 105 via the Internet 102. The web server would be the gateway to the Internet for a number of server based web applications 106a and 106b etc. The Internet Client could be a personal computer running browser software, a Wireless Application Protocol (WAP) mobile phone,

a Personal Information Manager (PIM), or other information-viewing device. For live connectivity these devices must support some form of network connection (e.g. by using the Java 1.1 Virtual Machine). The personal computer would typically be a workstation (e.g. PC compatible machine running a Windows™ operating system, Macintosh™ or Unix™

5   workstation); the browser software could for instance be Microsoft Internet Explorer™ version 5 (See www.Microsoft.com) or Netscape Navigator™ version 4 (See www.netscape.com).

In general the web server 105 would control the communication with the client using

10   standard HTTP 109. Typically the ALC would use HTTP for the initial communication but could then switch to using alternative communication protocols, such as TCP/IP or UDP, which better suit the application. In this way the ALC and the SE set up their own communication link 110.

15   One of the unique features of the ALC is that the Java Applet is small enough to be downloaded and configured on the browser each time the user starts a session with the application. This then gives rich client functionality without the need for add-ins and complex installation procedures.

20   Once the ALC is running on the client it creates an XML Database (inside the ALC space) on the desktop which not only stores all the data immediately required by the client but also has all the configuration information on how the interface should look and what the user preferences are.

25   **ALC - SE Interaction**
The key to the uniqueness of the ALC is how it interacts with the SE. The SE is a server-side layer that serves the ALC with the configuration, data and updates.
As can be seen in **Figure 2**, it is always the ALC that initiates the communication by requesting a page from the web server that contains the ALC applet 201.

The web server will pass the request on to the SE, which will return a page to the client containing the ALC applet. The ALC is then initiated in the client browser and from then on establishes and maintains communications with the SE. To allow the user interface to be

5   personalised, the user must make themselves known to the system and will therefore be allocated a unique key that can be used to reference personal preferences. Upon request from the ALC the SE amalgamates XML from a number of sources to send back to the ALC 204. Typically the SE could get information from the following:

10   ▪ General ALC configuration XML;
     ▪ User specific interface preferences;
     ▪ Web Application data, from potentially many applications.

The SE can be configured to retrieve this information in a number of different ways,

15   including standard HTTP calls, SOAP, JDBC, Socket, or JMS. The ALC will use this data to render a display to the user 205. The user can then interact with the data in a familiar desktop environment. As the user interacts with the application the ALC may contact the SE for additional data or it may initiate an update of the data.

20   When a user does something that requires additional data a chain of events takes place as described in **Figure 3**. The important step is that the SE can be configured to be an intelligent cache of web application data 304, thereby reducing the load on the web application itself. An example of this would be a news server. If the SE already has the requested news article then it doesn't need to interrogate the web application. The SE can

25   also be configured to only return changed data therefore reducing the amount of data sent 305. The SE uses a timestamp mechanism to ensure that a client connection only receives "new" data changes.

ALC Configuration

The ALC is totally configured with the use of industry standard XML. There are a number of aspects to the configuration:

- Interface definition
- Data Definition
- User Preference Definition
- Communications Definition

Referring to **Figure 4**, the interface definition 401 defines how the application will be rendered by the client in terms of windows, controls, images etc. This is initially defined for the application but then may be modified to take account of the user's specific configuration. The data configuration 402 defines the data to be served to the client. This is the XML definition of data that will be stored in the data section of the XML Desktop database. (The User Preferences 403, allow user specific requirements to be created and stored in the SE. These can then be used the next time the user logs on.

The Communication Definition will define how the ALC should interface with the SE in terms of the underlying protocol. Typically this would be HTTP Streaming or Polling.

E.   AltioLive – Synchronisation Engine (SE) Details

The Synchronisation Engine is a software application that can co-ordinate and control the interaction between multiple clients, multiple web applications or server data sources and multiple communication protocol types, while still being configurable via industry standard XML.

Some unique features of the SE are:

- The ability to interface with existing web applications using a variety of protocols, including conventional HTTP, JDBC, SOAP, Socket, and JMS;
- In built scalability to support large numbers of simultaneous clients and to keep them synchronised with each other and the web application;
5 - In built fault tolerance through parallel clustering of SEs;
- The ability to plug together multiple SEs and allow them to discover each other and their data sources;
- The ability to decouple the refresh rates of the web applications and clients;
- The ability to communicate with a number of different client types, for example
10 Internet browser, WAP phone, Personal Information Manager;
- Ability to pull together multiple web services;
- The ability to transport XML data elements to targeted destinations.

An SE is composed of a number of components:

15

- A network server machine;
- A server operating system;
- A communications protocol stack;
- A Java™ virtual machine;
20 - The SE Java servlet control programme,

These components are further described below:

The SE would be located on a network server machine. Typically there would be a CPU,
25 random access memory, permanent disk storage, ports for communicating with other servers on the local network, a manual input device and a display unit. Once the server was set up it would generally be administered remotely via the comms ports, in which case the keyboard and VDU might be removed.

The server is designed to run an operating system that is stored on permanent disk and is loaded into memory at start-up. The operating system would typically be Windows NT™ or Unix™ or Linux™, but could be any operating system that can support Java™ Servlets. To communicate over the local / wide area network the server would require a communications

5   protocol stack such as TCP/IP.

The software to control the SE would also be stored on permanent disk within the server and would also be loaded into memory as required. All the configuration information for the SE controlling software is stored as XML.

10

Communications

A key aspect of the SE is communications. The SE is able to communicate with back-end applications, with front-end clients and with other SEs. It can communicate at a number of different levels and in a number of different modes. At the lowest level the SE controlling

15   software will need to know the network protocol by which it needs to talk to each client and application. For communication to backend applications this could be HTTP, JMS, SOAP, Socket, or JDBC. Communication with the clients will typically be across the Internet and would therefore be over HTTP, but in some scenarios could be over TCP/IP or UDP.

The SE will need to know the network address of each communications partner, for

20   TCP/IP this would be the IP Address or the server name which this then translated into a TCP/IP address.

SE Configuration

The SE would be configured from a number of XML configuration definitions. Referring to

25   **Figure 4**, the server configuration 401 defines what data pools are going to be available and the definitions of the service functions required to interact with the web applications that are enabled through the SE. The server configuration also defines the architecture of the SE in terms of clustering, security and communications.

The data configuration 402 defines the data to be served to the client. This is the XML definition of data that will be stored in each of the data pools. Each data pool is a transaction log, which tracks changes to the data set(s) held in the client.

5   The application configuration 403 defines how the application will be rendered by the client in terms of forms, screen, images etc. This is initially defined for the application but then may be modified to take account of the users specific configuration.

The application default 404 will store system defaults such as refresh rates and other variables. These will be set at start up.

10

Data Interaction

Once communications are established between the SE, clients and web applications, the SE will need to transfer data between them. This can be by a number of methods:

- Push – This is where the SE propagates data to all parties that require it. This is used
15   to keep the clients in sync with web application data changes. Web applications can also be set up to push data to the SE.

- Pull – This is where the SE polls for the latest data, on a time basis or on user instigation. The client can also pull data from the SE. An example of a "Pull" is how changes to data on the web application are propagated back to the web client via the
20   SE. Typically a web page is only refreshed when the user requests one and when it is refreshed all the data is transferred from the server even if it hasn't changed. The SE introduces a number of alternatives depending on the application requirements. The SE can be configured to check for changed data at different time intervals 404. If the SE detects a change 408 independently of a RE (Rendering Engine/Layer)
25   request 402, then it can either cache the change or push it to all clients who currently require it. Changes would only be able to be pushed to the RE if a suitable communications protocol was in place such as HTTP Streaming, TCP/IP or UDP.

Simple Configuration

The first embodiment of this invention, Synchronisation Engine Simple Configuration, is depicted in **Figure 5**. This figure shows the simplest configuration consisting of one SE server 504 which would be connected to a number of applications 507 and optionally the
5   web server 508. It would also be connected to the Internet 503.

Complex Configuration

Figure 6 demonstrates how multiple versions of the SE can be pieced together to form a
10 scalable and fault tolerant network. This can be done without any code changes and with minimal configuration changes and the configuration that is required is all via XML. The SE system can be deployed many times to produce a hugely scaleable and fault tolerant system. In this embodiment there are two main SEs 601 & 602, these would act as dual redundant servers providing a high degree of fault tolerance. Each SE would have independent links to
15 the source applications 604. The SE invention embodies code that will allow these servers to keep themselves fully synchronised 603.

This approach provides a large degree of scalability by allowing the SEs to be setup in a snowflake configuration. The two main SEs, 601 & 602 would act as the hub and would
20 then be connected to other SEs 605 & 606 etc rather than directly to clients. Each one of these satellite SEs could then be linked to a large number of clients. The SE invention incorporates the propagation logic required to service this complex snowflake configuration.

More SEs could be setup in parallel to provide additional resilience and scalability. Also the
25 snowflake configuration could be extended to have many layers thereby giving near infinite scalability. Although in the above-described embodiment, the client was based on a web browser, it would be possible to have other types of user interface such as a Wireless Application Protocol (WAP) phone or a Personal Information Manager (PIM). The SE can

35

also be split over many Central Processing Units (CPUs) within the same server or over many servers.

F.   Deploying AltioLive

5

The generic applet must be configured so that the XML database layer on the client displays the correct data for a particular user, in the correct format and with window controls appropriate to that user, and in an optimal manner for the display capabilities of different client devices. The APS has responsibility for configuring the generic applet. This is achieved
10   via the XML Deployment Package (XDP). XDPs contain the definitions for the following AltioLive elements:

1. Service Functions – define the data (XML data & related Image data) available to users from the Web Application and the mechanisms available to update the Web Application.

15

2. Data Pools – define the mechanisms for live distribution of XML data updates to users.

3. AltioLive-Views – define the live interactive user-interface delivered through the AltioLive Client. There can be multiple views according to user roles.

20

4. Pervasive-Views – define the page-rendered user interface delivered through the Pervasive Client. There can be multiple views according to user roles and/or device capability.

5. Data-Views – define the data and access available through the Data Client for a DHTML
25   user interface.

6. Common elements – icons, desktops styles etc. for use across views within the XDP. XDPs can be developed as self-contained entities. Typically a self-contained XDP is used where the views are tied to a specific Web Application e.g. a contract resource application

could provide a requirement-view, a resourcing-view and a phone-based timesheet entry-view. Self-contained XDPs are also used in delivering third-party components (e.g. a Chat or e-mail component) that are effectively 'plugged-in' to application-specific XDPs. XDPs can also make use of definitions from other XDPs to define their views. Typically this is used

5   where the views merge XML data from many applications into a specific role-based view (e.g. a salesman sees a cross-system-view combining sales tracking data, customer management data and credit control; the sales director sees a view combining aggregated sales data, personnel monitoring and campaign performance data) or to use elements of a third-party

10

XDPs are developed within the AltioLive Development Edition (ADE) and then registered with the APS using the SE Admin Tool. This distributes the XDP across all Synchronization Engines in a cluster to make it available to users.

15   **G.   Security aspects of AltioLive**

AltioLive combines best-of-breed security technologies and practices to ensure a flexible, configurable and comprehensive security architecture for all applications. AltioLive is designed to integrate within existing security frameworks. It takes advantage of existing

20   session management and Secure Socket Layers and provides unique additional security features to offer a robust and highly secure environment for all systems.

User Authentication

25   In AltioLive, users do not directly log-on to the APS. AltioLive intentionally reuses the session object created in the web server/application server in order to let you take advantage of:

   ■   existing authentication policies for the application;

- best practices in user authentication;

- and the existing investment in technology infrastructure

When the end-user accesses the application through a host Web page, this page downloads

5   the appropriate Client from the same web server. The Client then contacts the APS, which uses the existing session object (on the server) to determine the specific View and XML data to be served to the user. Even if the URL of the host Web Page is known, the user cannot activate their connection with the APS unless they already have a valid session (including the necessary AltioLive specific variables) on the Web server. Using the existing session from

10   the Web server also prevents users from having to re-authenticate as they cross boundaries in a distributed system. AltioLive allows users to authenticate only once, regardless of the machine on which the application is deployed. To control who can connect to the APS, AltioLive leverages existing application servers' ability to enforce rules that govern whether a connection should be established based on client IP access or protocol. This allows

15   companies to decide if "Client" refers to AltioLive Client, AltioLive Pervasive Client, and the AltioLive Data Client.

Unique User Views and Data Pools

20   Individual or groups of users can be assigned different views/windows, which can be populated by data specific to each group's or individual user's need. Groups and/or individually registered users can subscribe to different data pools in the APS. Users only see the data to which they subscribe. In this way, AltioLive prevents unauthorized users from seeing sensitive data as well as allowing for unique user experiences/views. Through the

25   APS, system administrators can control which users see which data to ensure only the right data reaches the right users. For example, views could be configured for different roles in an organization (i.e. Salesperson, Sales Manager, Credit Controller etc.) that provide access to a number of backend systems – each role would have a view that restricts what functions are available and deliver only data that is relevant to their role. Specifying individual access levels

through the APS allows production systems to be up and running very quickly with minimal configuration and no coding.

Firewalls

5

Use of standard HTTP and HTTPS connections between the Client and the APS makes the technology transparent to most firewalls and does not require any relaxation of existing security policies. In highly secure environments where connections are restricted, the Client/APS connection is specific and easily added to the allowable list of sites. An

10 additional option for system administrators is to locate departmental slave (APS) servers that sit inside the firewall to support one or more users (who therefore no longer have to connect to the Internet). The slave server has a dedicated peer connection to its master APS that can be additionally secured according to company security policies

15 ## Connection Security

Not only is it important that users are properly authorized and authenticated when accessing data from the APS, but also that this data cannot be viewed while in transit. To avoid eavesdropping on connections between any connections to the APS, AltioLive offers users a

20 number of security features:

Client to/from APS

AltioLive includes support for a Secure Sockets Layer (SSL) and digital certificate

25 implementation that provide authentication and privacy via encryption in distributed APS applications. The Client/APS connection takes advantage of current SSL technology including 56 and 128 bit cipher strength connections. AltioLive is compatible with certificate authorities that support X.509v3 certificates. This means that developers are free to use the vast majority of certificate authorities available today.

39

In secure operation, the APS requires a digital certificate to be able to prove its identity to Clients and other servers. When a Client contacts the APS using SSL, they first request the server's digital certificate to prove its identity. Once the Client receives the server certificate,

5   identity is assured, and the Client and APS can confidently send data back and forth across the network. For highly secure environments, many installations need to also authenticate the Clients accessing the APS. For this reason, the APS SSL installation can be configured to also provide authentication for Clients and Web browser. This is called a two-way authentication and incorporates the same digital certificate verification process. The major

10   difference for two-way authentication is that the digital certificates are located in both the Client and the APSs. Once certificates have been exchanged and mutually verified, both the Client and the APS can trust the identity of the other. Furthermore, once mutual authentication has been set up, a private channel can be created between the two hosts consisting of encrypted data. Many systems will also require that the data which the Client

15   receives from the APS and vice versa to be secure/encrypted. SSL allows both the APS and the Client to encrypt the data exchanged between them. Potential hackers acquiring the data in transit are therefore unable to understand it. Streaming data connections are designed to make efficient use of secure connections to reduce server load as opposed to regenerating cipher keys for every request. Use of compression techniques on the data stream further

20   enhances security while reducing the overhead of encryption on the overall system. Both the Client and the APS monitor connections to detect loss of connection. Clients' connections can be configured to either fail-over to another APS or require the user to re-establish their session through a login.

25   **APS to/from APS**

The APS can be clustered for load balancing in master/slave relationships (as well as peer-balanced relationships). These connections are effectively point-to-point connections that can use HTTP, HTTPS or socket connections to secure the interface. If the APSs are

distributed in geographically different areas (not in the same data center behind the same physical firewall, for example), the SSL protocol provides secure connections by allowing these APSs to connect over the Internet and authenticate each other's identity and by encrypting the data between the APSs. For security above SSL, hardware encryption can be used on the specific connection between servers. All APSs accessing the cluster must be pre-registered with compatible license keys, which prevent spoofing of any of the servers.

APS to/from backend Web Application

The backend Web Application and the APS are usually in a closed and physically secured environment to prevent direct access to backend data from any outside client device. For implementations where the system connects across the Internet to external Web Services, SSL and digital certificates provide the necessary authentication and data encryption to ensure that the Web Service is a proper entity and all data transmitted is only decrypted and viewed by the intended recipient.

Reliable Data Delivery

Data integrity and delivery is assured through a combination of HTTP generated exceptions and the APS/Client timestamp mechanism. All messages from the APS to the Client are time stamped and sent across the network using HTTP(S). Data from the APS is sent to the Client using HTTP. Should a loss of network connectivity or a temporary hardware failure cause any data not to be received by the Client, the built-in handshaking in the HTTP protocol generates an exception. Essentially, if a message cannot be sent from the APS, the TCP/IP protocol generates an exception. This is generated in the server's Java Virtual Machine (JVM) and the exception is handled by the APS. The APS closes the connection upon receiving an HTTP exception. The Client will automatically reestablish the connection with the APS (the reconnection strategy is configured through AltioLive Tools). When the connection is re-established, the Client requests all updates since the last timestamp that it

41

successfully received, guaranteeing delivery of all messages in the correct sequence. For streaming data to the browser, the Client is just listening for data updates. In the case where it does not receive any updates or a probe message in a configured interval, it will start its re-connection process to the APS. In the meantime, the APS will generate an exception when
5   the probe message cannot be sent.

Transaction Authentication

To prevent an unauthorized user submitting a transaction it is common to use forms of
10   transaction authentication. This takes the form of access security/signature software on the client (sometimes even coupled to physical security – e.g. card swipe, challenge/response devices etc.) can add a security signature to any transaction. The AltioLive Client can be configured to call-out to such applications before submitting transactions to the server to either append the signature or encrypt the whole transaction.

15

Use of shared workstations

Unlike many solutions, the Client does not store any information locally on the workstation (configuration, data or even images). This means that the user can safely access their
20   application from shared/networked platforms without worrying that their data is accessible by others when they have logged-off. This is a key issue in the access-anywhere features of Altio's technology. Standard session time-outs can be used on the server to time-out inactive sessions. The Client can also be configured to disconnect on logout and/or moving from the host page to prevent someone using the Back/Forward functions of the browser to
25   reactivate the application (if it hasn't already timed-out).

Data Available to the User

The data available within the XML Desktop and visible to the user is configured at design

time and enforced by the APS. Users cannot configure their Client to access additional data not presented to their specific View.

Protection for the User

5

For the end-user, the Client is implemented within the standard Java 1.1 Security model. AltioLive Clients therefore do not have access to local data or services on the end-user's system and developers cannot undertake bespoke development to override the functionality of the Client. As an option, the Client can be 'signed' to give end-users additional comfort
10   that the software can be trusted for use on the specific application.

A Complete Security Architecture

In addition to all of AltioLive's security features, companies must also ensure that they
15   secure the general environment in which AltioLive is deployed. Such precautions include:

• Securing the operating system: No security hole in the DNS, sendmail or any other potential entry points.

20   • Securing File System: No general access to sensitive configuration information

• Securing Network: Use of firewalls and VPN

• Securing Physical Environment of the Server(s): Only authorized personnel have physical
25   access to system

• Separating Development and Production Systems

• No Source Code Available on Production System

The combination of real desktop-like functionality in online systems that users can access from anywhere, and the Internet becoming a proven and reliable medium for mission-critical communication is now driving companies to rapidly build new systems online as well as migrate existing systems to pure Internet Protocol-based applications using software platforms like AltioLive. Companies deploying IP-based applications must take special care that implementations are done in a secure and highly reliable environment to protect the substantial capital commitment required to build or to migrate any application online. AltioLive is designed to take advantage of existing security frameworks, such as session management, user authentication and Secure Socket Layers, reliable data delivery and transaction authentication, and it provides additional unique built-in security functionality to offer the most robust and highly secure environments for developers and end-users at a fraction of the cost and time of alternative techniques.

H.   Benefits of using AltioLive

With its XML-based architecture, AltioLive eliminates all the challenges posed by HTML, JavaScript and Dynamic HTML in the development of Internet applications.

Specifically, AltioLive provides companies the following benefits:

• **Increased User Functionality:** AltioLive provides a client/server-like environment directly inside the browser, giving users the same power as they have in traditional offline client/server-based systems. With AltioLive users have all the data they need at their fingertips in real-time, allowing the functionality in the system to be richly interactive. Graphs, drag-and drop, sorting, resizing and dynamically changing data are all available to users right inside the browser.

- **Enable Data delivery Across Devices:** AltioLive allows end-users to access Web applications from any Internet enabled mobile device (including Palm, WinCE, WAP phones, and Web TV) – freeing them from the traditional PC-centric Internet experience. AltioLive applications can be deployed almost anywhere as they are based on pure XML and

5   can be 'tuned' to meet people's various needs.

- **Savings on System Development:** Using the AltioLive Designer tool, developers drag and drop lists, graphs, buttons, images, and design windows with the use of their mouse. There is no coding required to develop the front-end, and only minimal coding for the

10   connection between the front-end and the back-end. This frees up precious programming resources to focus on building business logic rather than tedious and error-prone HTML. AltioLive significantly shortens development cycles, to enable faster time-to-market with considerable cost savings. Companies using AltioLive for large development projects have seen as much as 90% time and cost savings, equivalent to more than a million dollars in

15   upfront cost.

- **Savings on System Maintenance:** Since there is no code to be written for the front-end by the developers, AltioLive eliminates the chance of bugs through human error. Changes to the front-end can be implemented immediately using the AltioLive Designer tool.

20   Connections from the client to back-end servers are also handled by AltioLive and while some of the integration requires actual code from developers, most of the tasks are automated by AltioLive. AltioLive reduces bugs and eliminates the need to have significant IT resources or consultants on hand whenever changes have to be implemented.

25   - **Savings on Bandwidth:** AltioLive pushes data across the network only when the data actually changes. Even then, it only sends across the data that needs to be updated, not entire pages, tables, graphs or windows. Consequently, AltioLive-based systems substantially reduce the bandwidth used, saving companies substantial money on their bandwidth charges.

• **Fast Performance:** AltioLive-based systems deliver the initial data set in compressed form to the client. Users immediately see any updates to the data as it changes from the back-end as well as when they make any modifications themselves through the client. Most of the data will initially not travel over any network to a database for processing, but instead reside on
5   the client available to the user. The proximity of the data to the user gives faster performance than most client/server-based systems; it performs like a thick client.

• **Ability to Prototype New Features:** With AltioLive it is easy to develop and prototype new features and make incremental changes. There is no coding involved when using the
10   AltioLive Designer tool with XML-based Web services. The user interface, the logic between the different windows and the connection to the back-end servers can be configured by the business person who will actually be using the system.

• **Customized and role-based views:** Multiple views can be easily created for an
15   application (e.g. basic vs. advanced or manager vs. employee) and be automatically distributed by individual or role.

• **White Label Applications:** The look and feel of an AltioLive application is determined by 'skins' that allow very fine control (at the pixel level) of how applications look in the screen.
20   The AltioLive Designer tool can tailor applications for a number of customers (i.e. incorporate their logos, borders etc.) to provide users with a uniquely branded experience.

• **Secure Systems from Hackers:** AltioLive provides secure communication through standard SSL (Secure Sockets Layer) and HTTPS with the Web server or application server
25   doing the encryption and decryption in the server, and the browser doing the same on the client side. Additionally, the AltioLive applet runs in a carefully protected 'sand box' environment on the browser, preventing any ability to read/write to the hard disk drive on the computer. AltioLive-based applications can be used from public locations, such as net

46

cafes, libraries etc. because it is as harmless as a plain HTML page and leaves no data cached behind.

• **Less Load/Stress on Back-end Servers:** Changes to data in the database are pushed out
5   to the client in real-time, but only when changes occur and only the changed data is updated (not the whole window). This eliminates the need for users to constantly refresh the screen and make calls to the database on the back-end, which today causes heavy load when numerous users access the system. As users modify the data and submit new data back to the database, system administrators use the AltioLive Application Manager to configure how
10   often this data is sent back to the database. This effectively allows system administrators to tune the system for optimal performance. Companies using AltioLive have reduced server load by as much as 90%. Since AltioLive decreases the stress on the back-end servers, implementations require less servers, or at least less CPU-intensive servers, which reduces hardware costs by thousands of dollars. Additionally, as most software Licenses are based on
15   hardware CPUs, significant savings come from reduced software License fees. With certain enterprise software License fees approaching $75-100,000 per CPU, a reduction of even one or two CPUs could save a lot of money.

• **Ability to Implement Changes Quickly:** AltioLive allows companies to instantly make
20   changes, small or large, with virtually no coding on the client and only limited coding on the connection from the client to the server. This ability allows companies to adjust their systems on demand, rather than leave them with less than adequate functionality as business requirements change.

25

I.   **Glossary**

Altio Presentation Server (APS) - The operational platform for delivering live XML data to users through the Altio clients.

AltioLive Client (ALC) - Client that delivers a windows style live interface within a browser.

AltioLive Pervasive Client (APC) - Multi-device client that renders onto devices such as WAP, Web TV, PDAs.

AltioLive Data Client (ADC) - Data-only client (i.e. no presentation) for use with dynamic HTML user interfaces.

A 'client device' is any device which can be controlled over a network and has some means by which a human can view and/or interact with it. It could be as simple as a network controllable light bulb and switch or more complex, such as screen & keyboard based personal computers. Software on the client device can be embedded or installed onto the device or delivered on demand to the device over a network.

A 'data source' is any source of data potentially relevant to a network based application and includes basic data feeds (e.g. news headlines, equity quotes, sensor measurements) and more complex enterprise applications (e.g. Client Relationship Management, Human Resources, Enterprise Resource Planning and Accounting systems). Any 'data source' can also be independently or simultaneously a 'data receiver' and accept structured data and commands from the presentation server.

Domain Name System (DNS) - The Domain Name System (DNS) is an Internet directory service. DNS is used to translate between domain names and IP addresses, and to control Internet email delivery. Most Internet services rely on DNS to work. If DNS fails, web sites cannot be located and email delivery stalls.

16.   The presentation server of Claim 1 which insulates the client device from needing to know the quantity or configuration of multiple data sources to which the presentation server is connected, by aggregating data from multiple data sources into a single data stream for each client device.

5   17.   The presentation server of Claim 1 which maintains a data stream per client device in which the data is pushed over a persistent connection to the client device over the network as soon as it is available.

18.   The presentation server of Claim 1 which maintains a data stream per client device in

10   which the data is sent to the client device over the network in response to regular or ad-hoc polling requests from the client device.

19.   The presentation server of Claim 1 which compresses the data received from the data source prior to sending the information to the client device.

15

20.   The presentation server of Claim 1 which encrypts the data received from the data source prior to sending the information to the client device, thus insulating the data source from the enhanced security needed to allow a remote client device to run the network application over a public or insecure network.

20

21.   The presentation server of Claim 1 which decrypts the data received from the client device prior to sending the information to the data source, thus insulating the data source from the enhanced security needed to allow a remote client device to run the network application over a public or insecure network.

25

22.   The presentation server of Claim 1 which insulates the data source from needing to manage updates of the structured data held on the client device by regularly polling the data source for changes.

23.   The presentation server of Claim 1 which insulates the data source from needing to
        directly manage distribution of streaming data to multiple client devices, where the
        data source needs only to provide a single stream to a presentation server which in
        turn broadcasts the information in a scalable fashion to multiple (potentially more
5   than tens of thousands) of client devices that may in turn be connected to the
        presentation server at different performance rates over the network.

24.   The presentation server of Claim 1 which insulates the data source from needing to
        handle communications functions specific to different client devices by having a
10   generic communications capability (data receipt/acknowledgement/response)
        enabling it to communicate with multiple client browsers, multiple client device types
        and across multiple bearers.

25.   The presentation server of Claim 1 which insulates the data source from needing to
15   handle communications functions specific to different client devices by having a
        generic display capability by being able to download to the database layer one or
        more of the following, each configured appropriately for a destination display, device
        type and bearer: such as panels, windows, controls , data binding to the controls and
        event models.

20

26.   The presentation server of Claim 1 which can process a request by the client device
        for an update to a data source and return an indication of whether the update was
        successful as indicated by a response from the data source.

25 27. The presentation server of Claim 26 in which, if the update failed, the data source
        may optionally provide further information about the reasons for the failure in a way
        that can be displayed or interpreted by the client device.

99

28.　The presentation server of Claim 1 which can process a request by the client device for an update to a data source and return an indication of whether the update was successful even if the data source does not directly respond with a success or failure indication.

5

29.　The presentation server of Claim 28 which can be configured to automatically generate responses to the client device on behalf of the data source.

30.　The presentation server of Claim 28 which can process a request by the client device

10　for a single transaction update to multiple data sources and return an indication of whether the update was successful.

31.　The presentation server of Claim 30 which can process a request by the client device for a single transaction update to multiple data sources in such a way that all data

15　source updates are only confirmed once each data source has preliminarily indicated that the update would succeed.

32.　The presentation server of Claim 31 in which failure of any one of the transaction updates causes all the transaction updates to be cancelled and a failure response to be

20　returned to the client device that requested the transaction.

33.　The presentation server of Claim 31 in which a preliminary validation of all transactions causes all transactions to be confirmed and returns a success indication to the client device that requested the transaction.

25

34.　The presentation server of Claim 1 which returns the message that results in the user interface at the client device being updated at the same time as it issues a message that results in the user interface of different client devices being updated in the same manner.

35. The presentation server of Claim 1 which can update marginal portions of the database layer in a client device by sending, instead of an entire structured data tree hierarchy, a key which identifies the part of the structured data tree which has changed.

5

36. The presentation server of Claim 1 which can update a database layer by altering, in real time, configuration data held in the database layer which defines how the structured data can be interacted with from within the user interface.

10 37. The presentation server of Claim 1 which can prioritise updates to a client device that is returning on-line after an absence.

38. The presentation server of Claim 1 which can update the database layer by re-initialising all of the structured data on a client device that is returning on-line after an absence by applying rules to determine if re-initialisation is appropriate.

15

39. The presentation server of Claim 1 which stores updated structured data in a transaction log record from which a client can pull updates.

20 40. The presentation server of Claim 1 which stores updated data in a transaction log record and pushes updates as appropriate to the client device.

41. The presentation server of Claim 1 which is able to send updating data to different client devices at different times or frequencies.

25

42. The presentation server of Claim 1 which can be configured to skip unnecessary intermediate updates to streaming data for devices whose connection or client device is too inefficient to receive and process every update.

43.   The presentation server of Claim 1 which can maintain a statefull connection with the client device.

44.   The presentation server of Claim 43 which maintains a statefull connection with the client device by routing a unique, incrementing number to the client device, which the presentation server logs in relation to all data updates, so that when the client device returns on-line from an absence, the client device sends the latest number it received back to the presentation server so that the presentation server can determine what updates the client device did not receive and therefore needs to be sent.

45.   The presentation server of Claim 1 which is sent by the client device any parameters that need to be persistent and stores these parameters or routes them to storage for later retrieval.

46.   The presentation server of Claim 1 which provides scalability in the number of client devices a network based application can communicate with by being itself scalable with:

(a)   multiple CPUs on a machine;

(b)   multiple CPUs on multiple machines in a parallel or in a parent/child arrangement;

and managing the message flows from each CPU to the applicable client devices.

47.   The presentation server of Claim 1 which provides fault tolerance by being itself scalable with:

(a)   multiple CPUs on a machine;

(b)   multiple CPUs on multiple machines in a parallel or in a parent/child arrangement, providing a mechanism to automatically compensate for failures of instances on one or more CPUs by reconfiguring the network configuration at run-
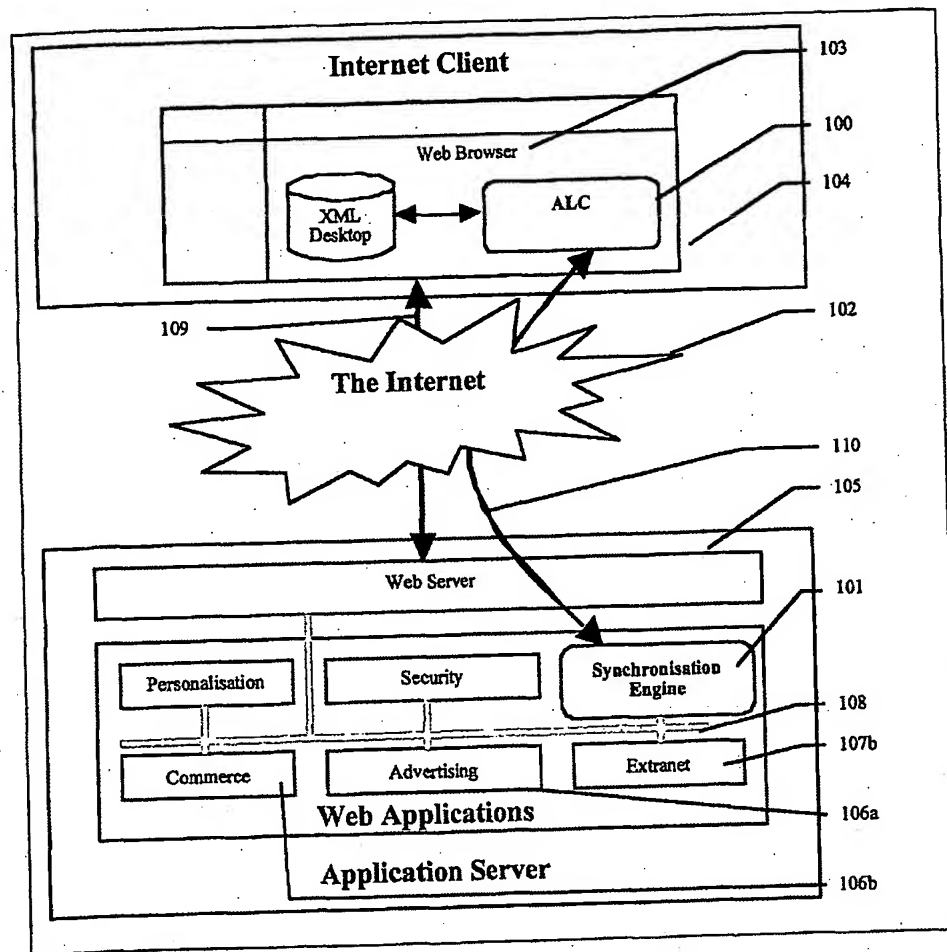
time and optionally re-establish connectivity for each orphaned client device to an alternate presentation server

48.   The presentation server of Claim 1, forming part of a network of several geographically separated presentation servers, some or all of which operate as edge servers.

49.   The presentation server of Claim 1 forming part of a network of several geographically separated machines organised into a topology which is appropriate to the physical arrangement of sites in a business running the network based application.

50.   The presentation server of Claim 1 which can combine structured data from several different server based applications for a given client device.

51.   The presentation server of Claim 1 which can determine automatically and in real time, when a client device requests several images, whether it should reduce bandwidth consumption by placing all of the images into a single, composite image file, each image sharing data which is not image specific.

52.   The presentation server of Claim 1 which modifies the data it sends to a client in dependence on information sent to it from the client which relates to the display limitations of the client device.

53.   The presentation server of Claim 1 which uses a session object associated with an applet to determine the specific display parameters and structured data to be served to the client device.

54.   The presentation server of Claim 1 in which the data source is a web server, a back
       end application or a web service.

55.   A web application which, when invoked or run, causes a presentation server defined
5   in any preceding Claims 1 – 54 to be run.

56.   A hardware server, which hosts a presentation server defined in any preceding
       Claims 1 – 54.

Figure 1

Overview of the components of the ALC

Figure 2

Initiate Client Communication

Request web page containing ALC applet	201
↓	
Initiate the ALC in Web Browser	202
↓	
ALC calls Synchronisation Engine (SE) via default communication protocol (HTTP).	203
↓	
SE provides initial client configuration & data	204
↓	
ALC renders initial user display.	205
↓	
ALC renegotiates communications	206
↓	
ALC contacts server for data updates or refresh	207
↓	
SE processes updates or provides refreshed data	208
↓	
ALC re-renders display	209

Figure 3

User Requests Data
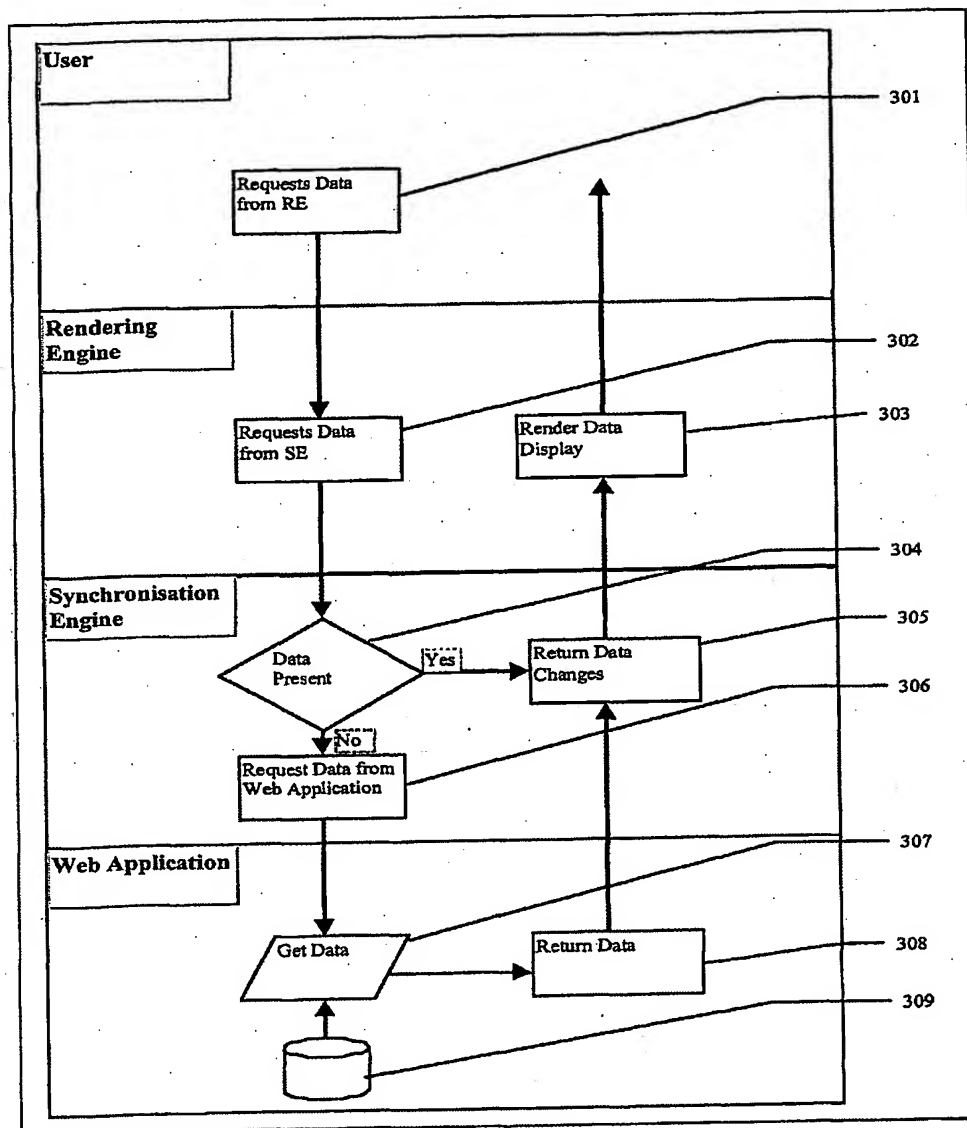
Figure 4

AltioLive Client Configuration

- 401

- 402

Server Configuration:
- Data pools
- Service functions
- Architecture

Data Configuration:
- Definitions of the data to be served to the client
-

Synchronisation Engine

- 403

- 404

Application Configuration:
- Definition of the application interface

Application Default:
- System defaults for the SE
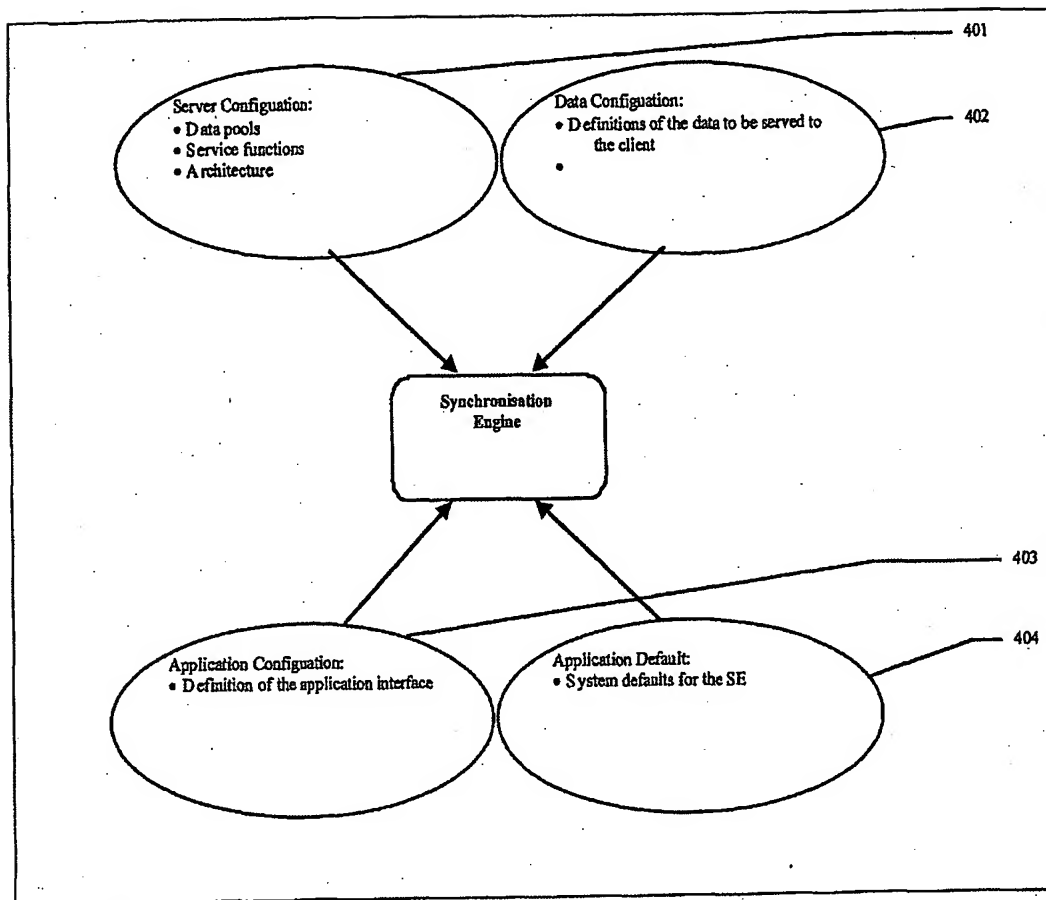
Figure 5

Synchronisation Engine Simple Configuration
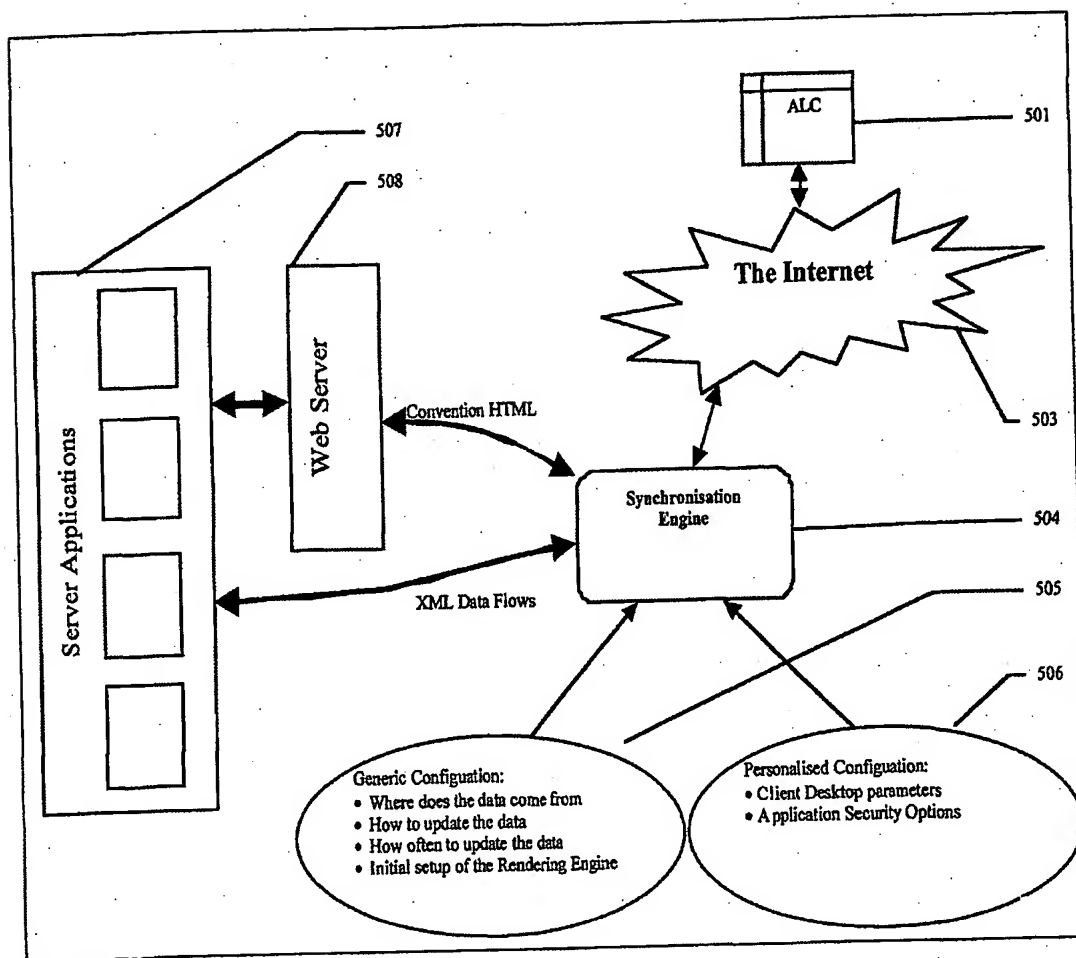
Figure 6

Synchronisation Engine Complex Configuration
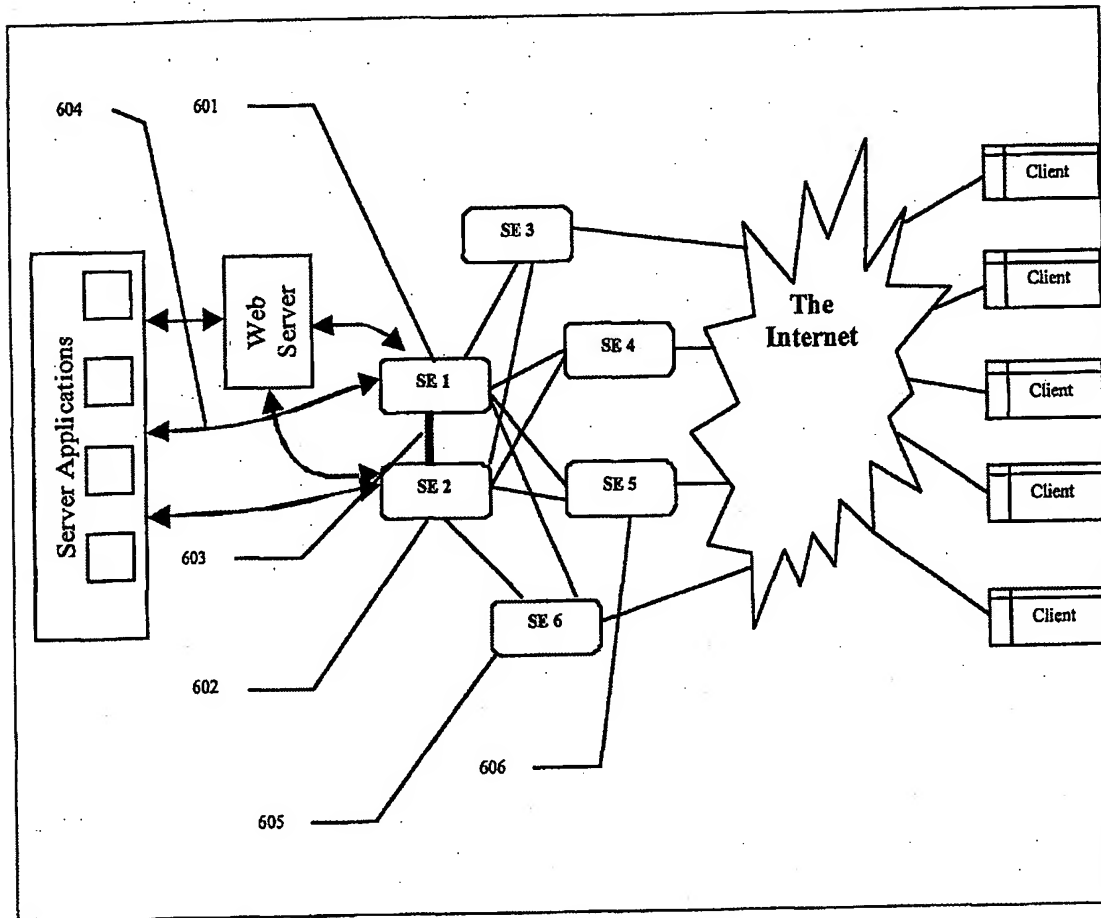
Figure 7

The AltioLive Deployment Architecture

Architecture
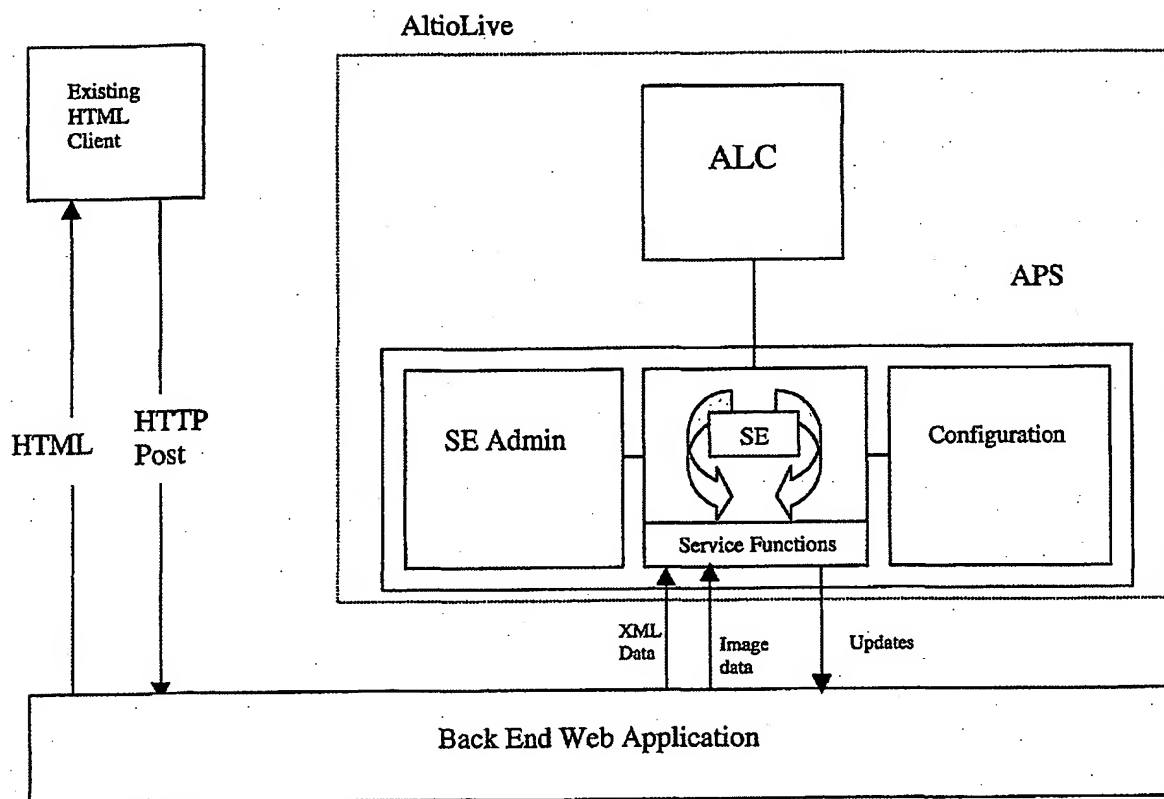
Figure 8

The Synchronization Engine serves the initial data according to the information associated with the View (shown below).

Figure 9

An example Service Function.

Figure 10

This screen view shows the datapool, which is used to get updates of items which the user has placed on their watch list.

Figure 11

A NEW_SALE Service Function.

```
┌────────────────────────────────────────────────────────────────────────┐
│ 🗑 HTTP Service NEW_SALE                                        🔽 ⊠     │
├────────────────────────────────────────────────────────────────────────┤
│  Service name:                 Document type:                            │
│  ┌──────────────────────┐      ┌────────┐    ☐ Return data              │
│  │ NEW_SALE             │      │ XML  🔽│                                │
│  └──────────────────────┘      └────────┘                               │
│  URL:                                                                    │
│  ┌────────────────────────────────────────────────────────────────────┐ │
│  │ http://${se.server.name}:${se.server.port}/${se.server.path}com.altio.altiodb.AltioDB │ │
│  └────────────────────────────────────────────────────────────────────┘ │
│  URL argument:                                                           │
│  ┌────────────────────────────────────────────────────┐    ☑ Add quote  │
│  │ APPID=ROS&ACTION=NEW&TARGET=/FOR_SALE&AL_NAME=SALE_ITEM& │           │
│  └────────────────────────────────────────────────────┘                 │
│  Transformation XSL:                                                     │
│  ┌────────────────────────────────────────────────────────────────────┐ │
│  │                                                                    │ │
│  └────────────────────────────────────────────────────────────────────┘ │
│  ┌─ Acknowledgement ──────────────────────────────────────────────────┐ │
│  │  Source        Text                                    Type         │ │
│  │  ┌─────────┐   ┌────────────────────────────────┐     ┌──────────┐  │ │
│  │  │ HEADER🔽│   │ Added record                   │     │SUCCESS 🔽│  │ │
│  │  └─────────┘   └────────────────────────────────┘     └──────────┘  │ │
│  │                  Source        Text                                 │ │
│  │  On success     ┌─────────┐   ┌────────────────────────────────┐    │ │
│  │                 │ DEFINE🔽│   │ Your for-sale item has been accepted│ │
│  │                 └─────────┘   └────────────────────────────────┘    │ │
│  │  On failure     ┌─────────┐   ┌────────────────────────────────┐    │ │
│  │                 │ DEFINE🔽│   │ There was a problem processing your request│ │
│  │                 └─────────┘   └────────────────────────────────┘    │ │
│  └────────────────────────────────────────────────────────────────────┘ │
│  ┌─ Parameter mapping ────────────────────────────────────────────────┐ │
│  │ ┌────────────────┬───────────────────────┬─────────────────────┐   │ │
│  │ │ Server         │ Client                │ Test              🔼│   │ │
│  │ ├────────────────┼───────────────────────┼─────────────────────┤   │ │
│  │ │ SELLER_ID      │ ${session logon name} │                     │   │ │
│  │ │ PROD           │ PROD                  │                     │   │ │
│  │ │ PROD_ID        │ PROD_ID               │                     │   │ │
│  │ │ QTY            │ QTY                   │                     │   │ │
│  │ │ PRICE          │ PRICE                 │                     │   │ │
│  │ │ CURRENCY       │ CURRENCY              │                   🔽│   │ │
│  │ │ SH_TPMS        │ SH_TPMS               │                     │   │ │
│  │ └────────────────┴───────────────────────┴─────────────────────┘   │ │
│  └────────────────────────────────────────────────────────────────────┘ │
├────────────────────────────────────────────────────────────────────────┤
│ 🗑 Enter Text                                                            │
├────────────────────────────────────────────────────────────────────────┤
│  ┌────────────────────────────────────────────────────────────────────┐ │
│  │ APPID=ROS&ACTION=NEW&TARGET=/FOR_SALE&AL_NAME=SALE_ITEM&GUID=SALE_ID │ │
│  │                                                                    │ │
│  │                                                                    │ │
│  └────────────────────────────────────────────────────────────────────┘ │
└────────────────────────────────────────────────────────────────────────┘
```
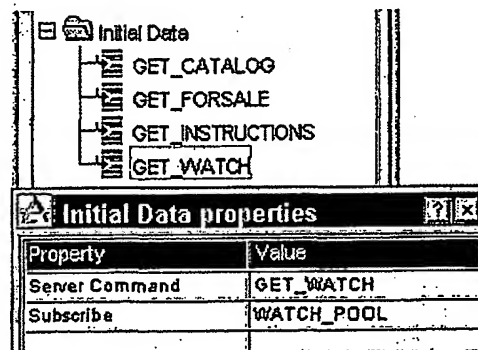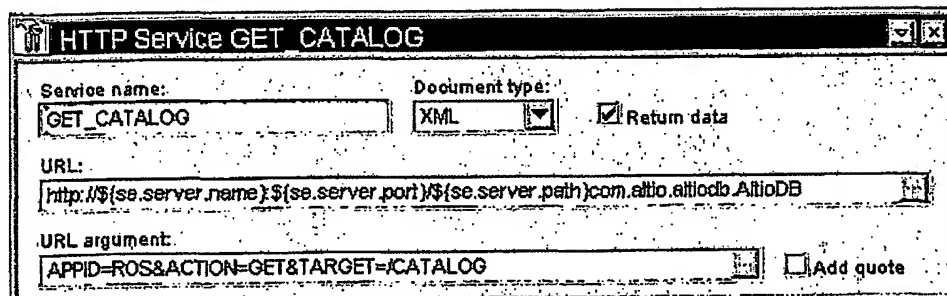
Figure 12

A NEW_BID Service Function

12/16

Figure 13

This element specifies the column of the "Select a product" list, including a GET_APPIMG image server command.

Figure 14

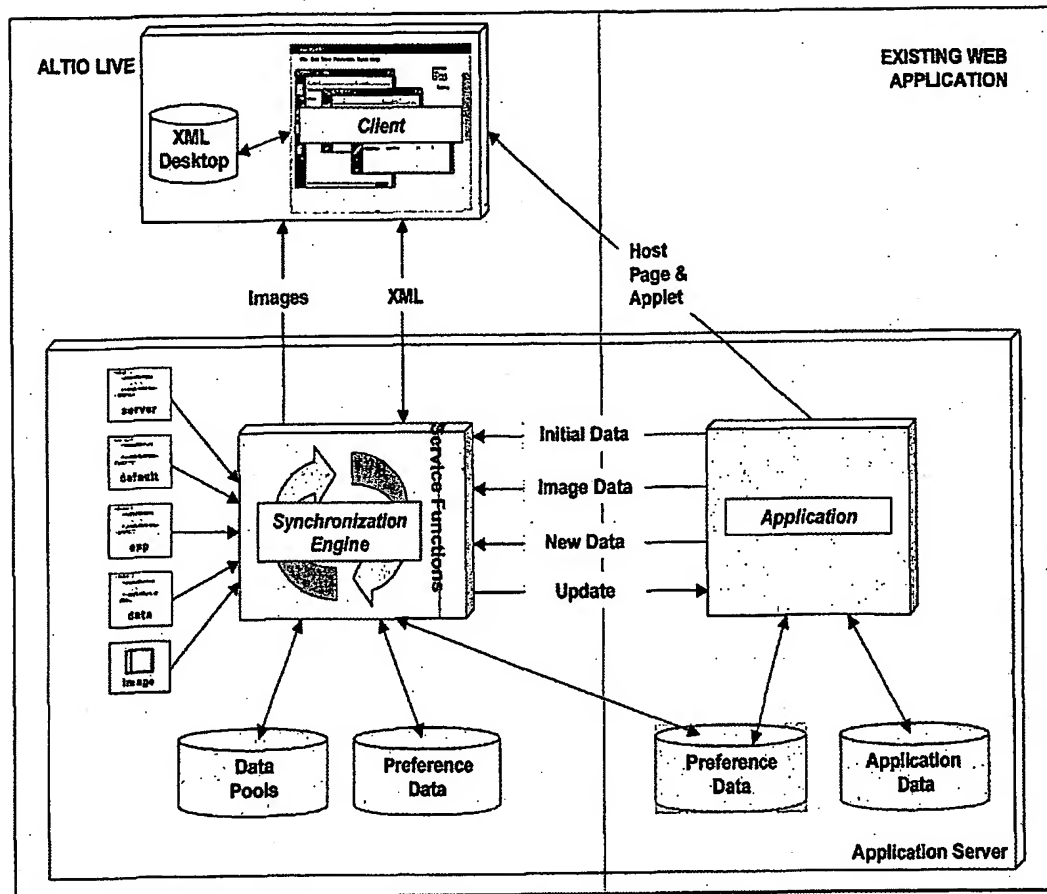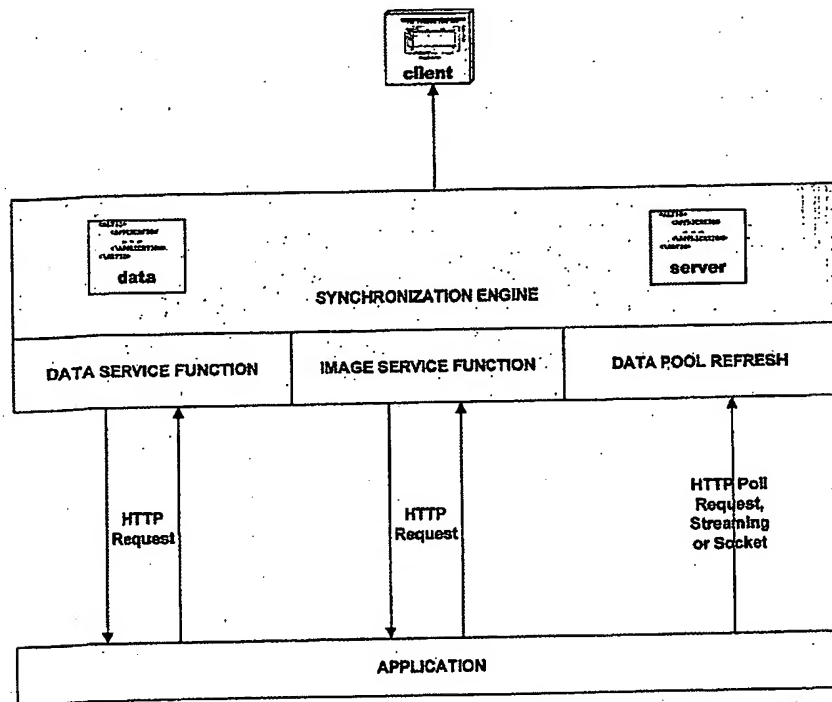The system architecture of an Altio Live application.

Figure 15

The interaction between the Sync Engine and the Application.

client

| data | | server |

SYNCHRONIZATION ENGINE

| DATA SERVICE FUNCTION | IMAGE SERVICE FUNCTION | DATA POOL REFRESH |

HTTP Request   HTTP Request   HTTP Poll Request, Streaming or Socket

APPLICATION

Key:

Icon	Name	Notes
	Sync Engine (Sync Engine)	The Altio Live Sync Engine running on an Application Server
client	Client	The Altio Live Client running within a browser.
server	Server Configuration	XML definition for the Sync Engine (Datapools, Service Functions, Architecture etc)
data	Data Configuration	XML definition for the data to be served to the Client.

15/16

Figure 16

When using Altio Live in Dynamic mode it is necessary to change the HTML to reflect the connection to the Sync Engine. Figure 16 shows a typical example of this integration.
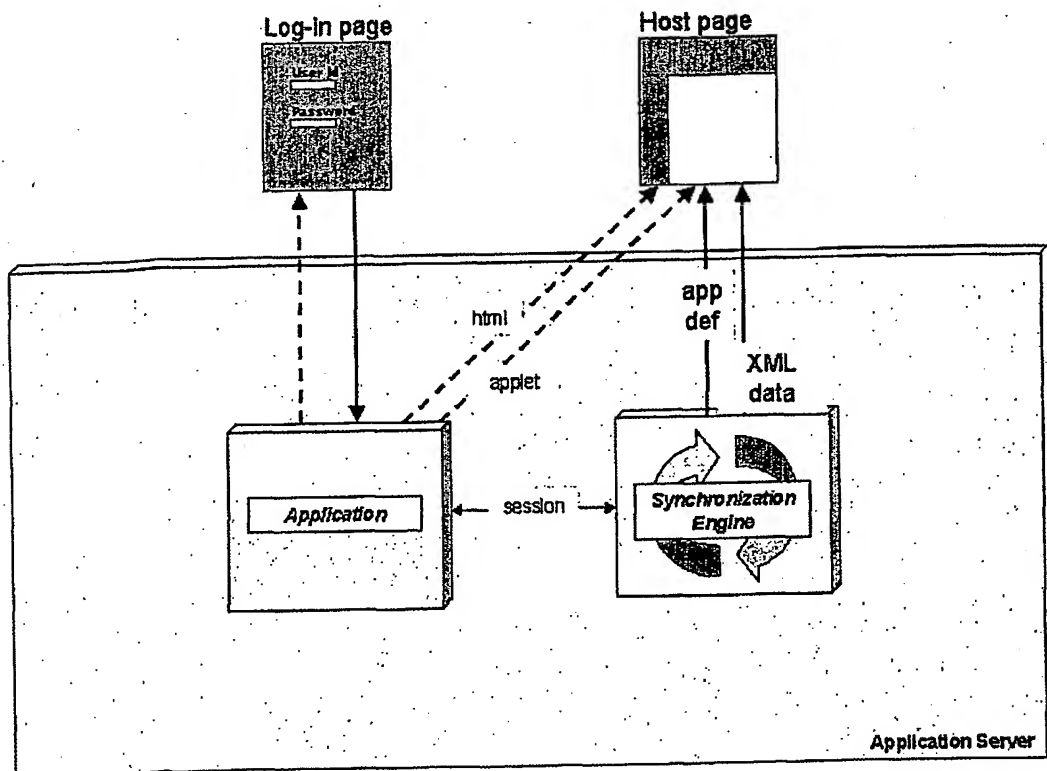
Figure 17

The Logon process used by the Altio Demonstration applications.